INTRODUCTION TO DATA STRUCTURE & ALGORITHM

Objective

- To map the complexities of Real-life problems to computer programming language as well as understand the concepts of
    - ✓Algorithm & Performance Measurement Parameters
    - ✓ADTs (Abstract Data Types)
    - ✓Data Structure
    - ✓Types of Data Structure and their applications

Course Structure

- Algorithms and their performance measuring parameters
  - ♠ Time Complexity
  - ♠ Space Complexity
  - ♠ Big-Oh (O) notation
- ADTs (Abstract Data Types)
- Definition and Classification of Data Structure
  - ♠ Primitive vs Non-primitive
  - ♠ Linear vs Non-linear
  - ♠ Homogeneous vs Non-homogeneous
  - ♠ Static vs Dynamic
- Types of Data Structure
  - ♠ Array
  - ♠ Linked List
  - ♠ Stack
  - ♠ Queue
  - ♠ Tree & Graph

Course Outcome

- Better understandability on how to solve various real-life problems.
- Increased coding capacity
- Better and efficient programming design

Algorithm

- **A good Program is a combination of both Data Structure & Algorithm.**
- **It’s a step-by-step process for solving a problem.**
- Must satisfy the following criteria:
  - ♣Every algorithm must take some inputs for processing.
  - ♣Every algorithm must produce some outputs after processing.
  - ♣The algorithm must terminate in a finite number of steps.
  - ♣The actions/steps of an algorithm must be clearly defined and well executed.
  - ♣The steps of an algorithm must maintain an order in which these steps are to be executed.
  - ♣Must use less memory space and take less time to perform.
  - ♣The algorithm must be easily understandable in such a way that its steps can be executed by pen & paper.

Efficiency of an Algorithm

○When we design an algorithm, we must watch out for these 2 issues:

♣**Validation of algorithm:-** The algorithm must give correct output for every input.

♣**Evaluating the complexity of an algorithm:-** By computing **Time Complexity** and **Space Complexity**.

○ **Time Complexity**:- Measures the total time taken by the program from its beginning upto the completion.

○ **Space Complexity**:- Measures the total memory space consumed by the program from its beginning upto the completion.

N.B.:- Space complexity is not as big issue as Time complexity because space can be reused, whereas time cannot.

Big-Oh (O) Notation

- **It denotes the maximum time taken by a program to complete.**
- General Format is **f(n) = efficiency denoted in O( )**

Examples

1.
```
for ( i =0; i<n; i++)
        System.out.println(i);
```
→ f(n) = O(n) because the loop will continue *n* times.

2.
```
for ( i =0; i<n; i=i+2)
        System.out.println(i);
```
→ f(n) = O(n/2) because the loop will continue *n/2* times.

3.
```
for ( i =0; i<n; i=i*2)
        System.out.println(i);
```
→ f(n) = O(log n) because the loop depends on the multiplication of 2.

4.
```
for ( i =0; i<n; i=i/2)
        System.out.println(i);
```
→ f(n) = O(log n) because the loop depends on the division of 2.

5.
```
for ( i =0; i<n; i++)
    for ( j=0;  j<n; j= j*2)
        System.out.println( i  +”    “+ j );
```
→ f(n) = O(n log n) because it’ll execute till *nlogn*.

6.
```
for ( i =0; i<n; i++)
    for ( j=0;  j<n; j++)
        System.out.println( i  +”    “+ j );
```
→ f(n) = $O(n^2)$ because the program will execute *n*n* times.

Data Structure

- **It’s a way of organizing the data items of a program to show various data items and their relationship.**

- **It deals with the study of how the data is organized in the memory, how efficiently the data can be retrieved and manipulated.**

- Used for better understanding between individual data elements for smooth execution of the program.

- Better organization of data means Better Efficient programs for implementing on a computer.

- A data structure requires following info:
  - Space for each data item it stores.
  - Time to perform a single basic operation.
  - Programming effort

Selection of Data Structure

- The selection of data structure complies with the following steps:
  - ♣ Analyze the problem to determine the data items need for a solution.
  - ♣ Determine the basic operations required for the data items to operate.
  - ♣ Select the data structure that best meets these requirements.

Classification of Data Structure

1. Primitive vs Non-primitive Data Structure
2. Linear vs Non-linear
3. Homogeneous vs Non-homogeneous
4. Static vs Dynamic

Primitive Data Structure	Non-Primitive Data Structure
These are the basic data structures that directly operate on the programming instruction.	These are derived from the primitive ones.
E.g. basic data types like, int, float, char, double, byte, short, long, Boolean, etc.	E.g. array, stack, queue, linked list

Linear Data Structure	Non-linear Data Structure
Here data items are arranged in a linear sequence.	Here data items are not arranged in linear sequence.
E.g. array, linked list, stack, queue	E.g. tree and graph

Static Data Structure	Dynamic Data Structure
Here memory size is fixed.	Here memory size is not fixed.
The structure is created at compile time.	The structure will be created at runtime according to the requirement.
E.g. array structure	E.g. linked list structure

Homogeneous Data Structure	Non-homogeneous Data Structure
Here data items are of same type.	Here data items are not of same type.
E.g. array	E.g. file

ADTs (Abstract Data Types)

- A data type that is defined entirely by a set of operations is referred as ADT.
- From ADT, users can only see the semantics and syntax of its operations, not all details.
- It encloses data and functions into a data type.
- It shows what a data type can do, not how it does.
- It is a collection of data and a set of rules that govern how the data will be manipulated.
- Examples of ADT- class, linked list, stack, queue, tree, graph, etc.
- Pros of ADT-
  - ♣Code is easier to understand.
  - ♣Implementation of ADTs can be changed without changing the total program.